A Framework for Online Conformance Checking

Burattin, Andrea; Carmona, Josep

Published in:
Business Process Management Workshops. BPM 2017.

Link to article, DOI:
10.1007/978-3-319-74030-0_12

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

A Framework for Online Conformance Checking

Andrea Burattin[1] and Josep Carmona[2]

[1] Technical University of Denmark, Denmark; University of Innsbruck, Austria
andbur@dtu.dk
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
jcarmona@cs.upc.edu

Abstract. Conformance checking – a branch of process mining – focuses on establishing to what extent actual executions of a process are in line with the expected behavior of a reference model. Current conformance checking techniques only allow for *a-posteriori* analysis: the amount of (non-)conformant behavior is quantified after the completion of the process instance. In this paper we propose a framework for *online conformance checking*: not only do we quantify (non-)conformant behavior as the execution is running, we also restrict the computation to constant time complexity per event analyzed, thus enabling the online analysis of a *stream* of events. The framework is instantiated with ideas coming from the theory of regions, and state similarity. An implementation is available in ProM and promising results have been obtained.

Keywords: online process mining, conformance checking, event stream.

1 Introduction

Process mining [1] represents an important research and industrial topic comprising the analysis of data regarding business processes in order to extract knowledge. Within process mining, different problems are typically identified and, in this paper, we focus on *conformance checking*. Conformance checking techniques, given as input a reference process model and an execution trace, compute the extent to which the executed actions conform the given model. Since most information systems allow for a certain amount of flexibility and deviations, conformance checking represents an extremely valuable tool.

All techniques available nowadays require a complete trace in order to calculate their conformance. From a business point of view, however, this represents an important limitation: if the trace is already finished, the countermeasures needed to fix the deviation can be implemented at a very late stage (i.e., when the process instance is already completed). In this paper, we drop such requirement and present a technique capable of computing the conformance for *running* process instances. Therefore, if a deviation from the reference behavior is observed, the system notices immediately the problem and allows for an immediate response. When these errors are accumulating, the "seriousness" of the process instance is raised, thus providing stronger alerts to the process administrator. In this paper, we focus on imperative process models, such as Petri nets.

In the context of this paper, with the term *online* we refer to the type of input of our technique: we assume to have an *event stream* which, basically, is a *data stream* of events. According to [14, 4, 5], a data stream consists of an unbounded sequence of data items which are generated at very high throughput. To cope with such data streams, in the literature, typically the following assumptions are made: *(i)* each item is assumed to contain just a small and fixed number of attributes; *(ii)* algorithms processing data streams should be able to process an infinite amount of data, without exceeding memory limits; *(iii)* the amount of memory available to an algorithm is considered finite, and typically much smaller than the data observed in a reasonable span of time; *(iv)* there is a small upper bound on the time allowed to process an item, e.g. algorithms have to scale linearly with the number of processed items: typically the algorithms work with one pass of the data; *(v)* stream "concepts" (i.e. models generating data) are assumed to be stationary or evolving. The literature reports several algorithms for the analysis of data stream [12, 4, 13]. However, typically these works cope with different problems, such as classification, frequency counting, time series analysis, and changes diagnosis (concept drift detection).

In the remainder of the paper, Sec. 2 presents the state of the art. Sec. 3 describes the technical details of our proposal, while Sec. 4 presents the implementation and performance results. Sec. 5 concludes the paper.

2 Related work

In [22], authors compared an event log with a Petri net to compute *fitness* and *appropriateness* measures. A different family of approaches relies on *alignments* [3]: in [2] the idea is to "align" a given trace with the most similar one that can be generated by the given model. Optimized versions have also been proposed [20, 6, 19]. All these techniques, however, cannot be applied in online settings since they require a complete trace. The main contribution of this paper is to drop this requirement, as we can compute the conformance during the execution of the process, and not *a-posteriori*.

Online process mining has also been investigated, but just concerning the control-flow discovery problem: algorithms generating Petri nets [7, 11, 23] as well as Declare models [9, 10, 16] have been proposed. These approaches, however, are not capable of checking the conformance. Mixed approaches to discovery and guarantee conformance values with one pass over data are available [15] but these still require a finite log with complete traces.

A related field of research is operational support. In this case, the system is capable of providing contextual information for running process instances, e.g. [17, 18] for the Declare context. However, as soon as a deviation is observed, corresponding Declare constraint are marked as permanently violated. In conformance checking, instead, the behavior might come back to a normal state.

3 Proposed Approach

Given a process model, we want to analyze an event stream, in order to detect (and notify to the process analyst) running cases that are deviating from the

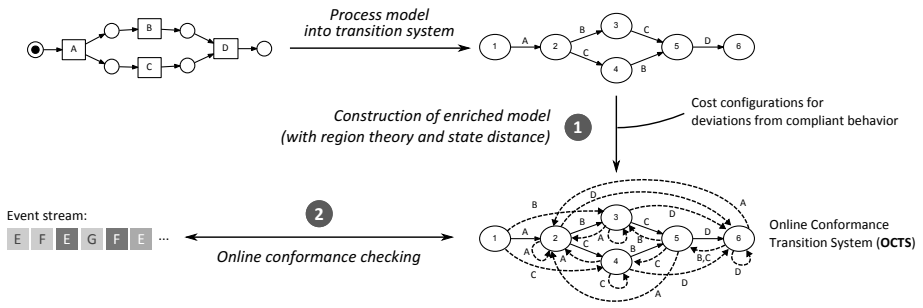
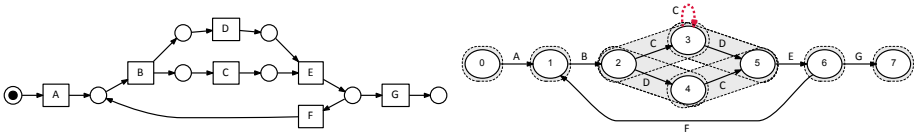


Fig. 1: Approach representation. Circled numbers represent the involved steps.

behavior prescribed by the model. We assume our input process model is represented as a Petri net [21] and we are going to leverage the notion of region theory and states similarity to achieve our goal.

The strictness of the rules governing the online scenario is playing a fundamental role in this case. In particular, most recent approaches for conformance checking on Petri nets are based on finding the *optimal alignment* between an observed trace and the *closest* possible trace allowed by the model. By moving into the online scenario, we know in advance the impossibility to achieve the same goal (i.e., find the optimal alignment). This is due to the impracticability of backtracking operations while analyzing an event of the stream. To avoid these operations, we devised a two-steps approach as depicted in Fig. 1: we first (i.e., offline, before the analysis) embed all computations in an augmented model, and then we perform the online analysis on such augmented model. The model extension has to provide information on how to deal with any uncompliant scenarios and these "wrong behaviors" are associated with costs larger than 0. Then, with such a model, it is possible to process the stream by analyzing one event at a time in constant time and, therefore, fulfilling the requirements for online algorithms. The online analysis consists of *replaying* the events of each trace of the stream on the model and accumulating the costs associated with each execution. All running process instances with costs larger than 0 are deviating from the reference model. Additionally, the larger the cost the more problematic is the process instance.

In online scenarios, it is not always relevant to mimic the concept of alignments. For example, in very critical environments, we might want to immediately raise alarms when deviations take place. In such cases, we need to extend the model with *sink states* collecting all deviations. We might also associate different costs to different sink connections, thus providing fine-grained alarm levels. In this paper, however, we would like to leverage some alignments concepts. In order to precompute sub-optimal solutions that are similar to optimal alignments, we will use both region theory and states similarity. For example, considering the model depicted in Fig. 2a, and corresponding reachability graph in Fig. 2b we can highlight the corresponding 8 minimal regions (in dashed line). Informally, a region (e.g., the set of states $\{2, 3\}$ in Fig. 2b), denotes a set of states for which



(a) Petri net model

(b) Reachability graph, minimal regions, and one "extension edge" (dotted red)

Fig. 2: A simple Petri net and the corresponding reachability graph.

arcs have an homogeneous relation. Each region corresponds to a place in the Petri net (e.g., the aforementioned region corresponds to place between transition B and D in Fig. 2a. Let's now assume the following trace $\langle A, B, C, C\rangle$. After executing $\langle A, B, C\rangle$ the trace reaches state 3. From this state, however, a new execution of C (which represents a deviation from reference behavior) should remain in state 3 since such state belongs to a region where C does not change the behavior of the model. Therefore, we extended our transition system with such self loop (highlighted in dotted red). Consider now a partial execution $\langle A, C\rangle$. After executing A we have uncompliant behavior and we would like to synchronize the execution with the model again. To do that, we should connect state 1 with a state which has a transition labeled C entering (i.e., 3 or 5). In this case, it is important to check the "similarity" of the assumptions of two target states. In this paper, we analyze these cases by checking the activities that lead to the possible states and considering the most likely configuration.

Concerning the computation of regions, please note this is a very expensive operation. However, the transition system we use as input is actually generated from a Petri net. And, in this case, each place of the Petri net defines a region. Therefore, given the set of reachable states and a place, all those states where the place has a token define the corresponding region.

The rest of the paper uses these preliminary definitions:

Definition 1 (Sequence). *Given the first n positive natural numbers $\mathbb{N}_n^+ = \{1, 2, \ldots, n\}$ and a target set A, a sequence σ is a function $\sigma : \mathbb{N}_n^+ \to A$. We say that σ* maps *indexes to the corresponding elements of A. For simplicity, in this text, we refer a sequence using its string interpretation: $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$, where $a_i = \sigma(i)$ and $a_i \in A$.*

We assume typical operators are available over sequences and behave as expected. For example, given a sequence $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$ and an element a, we have $a \in \sigma$ (reads "element a is contained in σ") if $\exists\, i \in \mathbb{N}_n^+$ such that $a_i = a$. In the context of this work, we refer to a trace as a sequence of events. Formally:

Definition 2 (Trace). *Given a set of activities A (e.g., the tasks of a process model), a trace T of length n is a sequence $T : \mathbb{N}_n^+ \to A$. Activities are grouped in the same trace when they are part of the same process instance.*

Please note that, in the online context, though we assume an infinite number of traces, the length of each of them is typically assumed finite.

Definition 3 (Event Stream). *Given the* event universe $\mathcal{E} = A \times \mathcal{C}$*, where* A *is the set of activities and* $\mathcal{C}$ *is the set of possible case ids, an event stream* Ψ *is an infinite sequence* $\Psi : \mathbb{N}^+ \to \mathcal{E}$*.*

3.1 Construction of the Enriched Model

The input of our approach is a Petri net:

Definition 4 (Petri net). *A* Petri net N *is a tuple* $N = (P, T, F)$ *where* P *is a set of places;* T *is a non-empty, finite, set of transitions, such that* $P \cup T = \emptyset$*; and* $F \subseteq (P \times T) \cup (T \times P)$ *is a flow relation.*

Given a Petri net $N = (P, T, F)$, a *marking* of N is a function $M : P \to \mathbb{N}_0$ mapping each places to the number of tokens it contains. The set of all possible markings is denoted with $\mathbb{M}$.

Definition 5 (Petri net system). *A* Petri net system *PS is a tuple* $PS = (P, T, F, m_0)$ *where* $N = (P, T, F)$ *represents a given a Petri net and* $m_0 \in \mathbb{M}$ *is the initial marking of* N*.*

As explained before, in order to compute in advance all possible configurations we may have to deal with, it is necessary to construct a behavioral model describing the different configurations and their interactions. To represent such model, we use a labeled transition system:

Definition 6 (Labeled transition system). *A* labeled transition system *TS is a tuple* $TS = (S, \Sigma, \delta)$ *where* S *is a finite set of states;* Σ *is a finite alphabet; and* $\delta \subseteq S \times \Sigma \times S$ *is a state transition relation.*

Converting a Petri net into a labeled transition system is a well-studied operation and, in particular, it results in the construction of the so-called *reachability graph*. The reachability graph is finite only if the starting Petri net is bounded, but algorithms which deal with unbounded cases to create a *coverability graph*[3] have been proposed. The idea is to map reachable markings of the Petri net system into states of the transition system (i.e., S represents the subset of $\mathbb{M}$ which is reachable). State transitions, in turn, connect different reachable markings and are labeled according to the Petri net transition leading to the target state. In order to compute online conformance checking, however, we need to extend the transition system definition in order to deal with initial state and costs. Therefore, we define:

Definition 7 (Extended transition system). *Given a transition system* $T = (S, \Sigma, \delta)$ *we define an* extended transition system *as* $T_{ext} = (S, \Sigma, \delta, w, s_0)$ *where* $w : \delta \to \mathbb{N}_0$ *is a cost function which associates transitions to a cost, and* $s_0 \in S$ *is an initial state.*

[3] The *coverability graph*, actually, does not represent a good transition system for conformance purposes, as it allows for more behavior with respect to the original Petri nets. Therefore, in this paper, we assume that the given Petri net is bounded. This assumption is typically fulfilled in many real-world applications.

Moreover, given a Petri net system $PS = (P, T, F, m_0)$, and the corresponding transition system $TS = (S, \Sigma, \delta)$ (e.g., the reachability graph) we can construct its extended transition system by considering all states, all transitions and all labels of the transition systems and setting all initial weights to 0 (i.e., $w = \{(d, 0) \mid d \in \delta\}$) and associating s_i to the state corresponding to m_0. On top of an (extended) transition system, it is possible to define the concept of *region*.

Definition 8. *Given a transition system $TS = (S, \Sigma, \delta)$, let $S' \subseteq S$ be a subset of states and $\sigma \in \Sigma$ be a letter of the alphabet. We define:*

$$\text{nocross}(\sigma, S') \equiv \exists (s_1, \sigma, s_2) \in \delta : s_1 \in S' \Leftrightarrow s_2 \in S'$$
$$\text{enter}(\sigma, S') \equiv \exists (s_1, \sigma, s_2) \in \delta : s_1 \notin S' \wedge s_2 \in S'$$
$$\text{exit}(\sigma, S') \equiv \exists (s_1, \sigma, s_2) \in \delta : s_1 \in S' \wedge s_2 \notin S'$$

Based on these conditions, we can define a region:

Definition 9 (Region). *Given a transition system $TS = (S, \Sigma, \delta)$, a set of states $R \subseteq S$ is called* region *if, for all $\sigma \in \Sigma$, both these conditions are fulfilled:*

- $\text{enter}(\sigma, R) \Rightarrow \neg\, \text{nocross}(\sigma, R) \wedge \neg\, \text{exit}(\sigma, R)$;
- $\text{exit}(\sigma, R) \Rightarrow \neg\, \text{nocross}(\sigma, R) \wedge \neg\, \text{enter}(\sigma, R)$.

Let R and R' be regions of TS. R is minimal *if there is no R' such that $R' \subset R$.*

Informally, a region is a subset of states where all transitions with the same label share the same enter/exit relationship. Algorithms for the identification of minimal regions have been proposed in the literature.

An extended transition system of a Petri net allows for the replay just of traces that conform the process and, to be general enough, we have to consider deviations. To do so, we add additional transitions to the system, associating them with costs larger than 0. In the end, the transition system has to allow the execution of all possible events from any given state. We call such transition system an *online conformance transition system* (OCTS):

Definition 10 (Online Conformance Transition System (OCTS)). *An extended transition system $T_{ext} = (S, \Sigma, \delta, w, s_i)$ is an OCTS if: $\forall s \in S, \forall \sigma \in \Sigma$ we have $|\delta(s, \sigma)| = 1$.*

An OCTS is always deterministic since, for each state, it has exactly one transition for each label in our alphabet of possible transitions. An OCTS can be used to *replay* traces:

Definition 11 (Replay). *Given an OCTS $O = (S, \Sigma, \delta, w, s_i)$ and a (partial) trace $t = \langle t_1, \ldots, t_n \rangle$, a replay of trace t in O is $R_t^O = \langle \delta_1, \ldots, \delta_n \rangle$ such that for all $(s_i^s, l_i, s_i^t) \in R_t^O$ we have $l_i = t_i$ (i.e., the transitions of the replay correspond to the activities of the trace) and $\forall j \in \{1, \ldots, n-1\} s_j^t = s_{j+1}^s$ (i.e., the target state of each transition corresponds to the source state of the following one). The cost of the replay R_t^O is: $cost\left(R_t^O\right) = \sum_{d \in R_t^O} w(d)$.*

Since OCTSs are deterministic, given an OCTS O and a trace t, the replay R_t^O is unique. Necessary additional properties of an OCTS O are: *(i)* given a conformant trace t, then $cost\left(R_t^O\right) = 0$; *(ii)* given a non-conformant trace t, then $cost\left(R_t^O\right) > 0$.

The way a transition system of a Petri net is extended into an OCTS represents how we are dealing with deviations. Different ways of dealing with deviations might be implemented in this framework and, in the rest of this section, we suggest one. Let's, for example, consider the process in Fig. 2a with its corresponding reachability graph (cf. Fig. 2b) and the scenario in which the replay reached state 1 (i.e., the trace $\langle A \rangle$ was observed). At this point, if the stream contains activity C, there are three choices: *(i)* to ignore C, or to execute C by assuming to go into state *(ii)* 3 or *(iii)* 5. In this case, we prefer to execute C to state 3 and this is due to the "contextual" information: considering the past histories of the traces leading to the two states, 3 is more similar to 1 with respect to 5. In some other situations, instead, we'll stay in the same state even though we did observe an unexpected activity. This option (i.e., what we indicate with option *(i)*) is considered if in any of the regions the current state belongs to there is a transition that is labeled as the observed activity and that does not cross the border of the region. The rationale is that the occurrence of activities that do not cross regions are not affecting the local state of the underlying system, and therefore it can be assumed that whilst an activity was observed, the system remains in the current state.

A formal representation of the enrichment approach is reported in Alg. 1. It takes as input an extended transition system (e.g., created starting from the reachability graph), and 3 cost parameters. The procedure is structured in 3 parts: the first (line 1) is in charge of setting the costs to 0 (i.e., correct behavior) for the transitions already in place. The second part (lines 2-5) adds to each state the possibility to replay any activity not belonging to the process alphabet (i.e., $*\backslash\Sigma$).The third part of the algorithm (lines 6-19) is in charge of extending the set of transitions to cope with deviations. The algorithm needs to process each state (line 6) in order to allow the execution of any event (line 8, which filter those not already in place). At this stage there are two options. The first case deals with transitions that are not crossing the regions the current state belongs to (lines 7 and 9-12). In this case, we just add a self-loop (line 10) and set the proper cost (line 11). If there's no transition with the same label in the region, then we first select the candidate states (line 13, those with and incoming transition labeled as the activity we're dealing with) and then we pick the candidate maximizing the cosine similarity of the vector representation of the candidate with the current state (line 14). Corresponding edges and costs (lines 15 and 16) are added to create the OCTS.

The vectorial representation of a state s, considering the target activity execution e, indicated as $vec(s, e)$, consists of an n-dimensional vector, where $n = |\Sigma|$ (i.e., the size of the alphabet, which is the number of different transition labels in the original Petri net) and where components correspond to letters of our alphabet. Each v_i, referring to letter Σ_i, is valued as follow:

- if $\Sigma_i = e$ (i.e., if the letter refers to the target activity), then the value is 0;

Algorithm 1: Enrich an extended transition system into an OCTS

Input: $T = (S, \Sigma, \delta, w, s_i)$: an extended transition system
c_s: cost of skipping the activity
c_j: cost of jumping to the next synchronous move
c_u: cost of activities not in the alphabet

▷ Set initial costs to 0 for all conformant transitions
foreach $d \in \delta$ **do** $w(d) \leftarrow 0$
▷ Add self loops for activities not in the alphabet
foreach $s \in S$ **do**
  $\delta \leftarrow \delta \cup (s, * \setminus \Sigma, s)$
  $w \leftarrow w \cup ((s, * \setminus \Sigma, s), c_u)$
end
▷ Add transitions to deal with deviations
foreach $s \in S$ **do**
  ▷ Construct the set of states sharing a region with s
  $R \leftarrow \bigcup_{S' \in Regions(T)} \{s' \in S \mid s \in S' \wedge s' \in S'\}$
  ▷ Consider all possible following activities except those allowed by the model
  foreach $e \in \Sigma \setminus \{e' \mid (s, e', s') \in \delta\}$ **do**
    ▷ Check if at least 1 transition labeled e connects 2 states in the regions of s
    if $e \in \{e' \mid (s^s, e', s^t) \in \delta \wedge s^s \in R \wedge s^t \in R\}$ **then**
      ▷ Option 1: Skipping the activity
      $\delta \leftarrow \delta \cup (s, e, s)$ ▷ Add a self loop when e is observed
      $w \leftarrow w \cup ((s, e, s), c_s)$ ▷ Add proper cost for the self loop
    else
      ▷ Option 2: Align to synchronous move
      $C \leftarrow \{s^t \mid (s^s, e, s^t) \in \delta\}$ ▷ Set of candidate states
      ▷ State maximizing cosine similarity with s. Details about vec on the text
      $s_{goal} \leftarrow \text{argmax}_{c \in C} \; vec(c, e) \cdot vec(s, e) / \|vec(c, e)\| \|vec(s, e)\|$
      $\delta \leftarrow \delta \cup (s, e, s_{goal})$
      $w \leftarrow w \cup ((s, e, s_{goal}), c_j)$
    end
  end
end

– if there exists a path from the start state to the current state s, containing label Σ_i, then the value is 1;
– in all other cases, the value is 0.

Considering again the example reported in Fig. 2b, given state 1 and activity C, we have the following representations:

$$vec(1, C) = [1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0] \qquad \begin{matrix} & A & B & C & D & E & F & G \\ vec(3, C) = & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ vec(5, C) = & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{matrix}$$

The cosine similarity between $vec(1, C)$ and $vec(3, C)$ is 0.71, whereas the similarity between $vec(1, C)$ and $vec(5, C)$ is 0.58. For this reason, state 3 is preferred. Therefore, the algorithm will introduce the edge with label C connecting state 1 and state 3.[4]

Mapping the construction of an OCTS to alignments, please note that edges with cost 0 correspond to synchronous moves. Self loops with cost larger than 0 correspond to log moves. Those edges with cost larger than 0 that are not self loops, correspond to model moves on silent actions (i.e., there's a change in the model state) followed by a synchronous move. By using such mapping an

[4] The OCTS of the system reported in Fig. 2b is available at `https://andrea.burattin.net/public-files/online-conformance/octs.pdf`.

Algorithm 2: Online conformance checking

Input: $O = (S, \Sigma, \delta, w, s_i)$: the OCTS of the reference model
Ψ: event stream
m: maximum number of parallel instances

1 $M_O : \mathcal{C} \to S \times \mathbb{N}^+ \times \mathbb{N}^+$ ▷ Hash map which, given a case id, returns a tuple with a pointer to the current state in O, the cost of the process instance so far, and the time of last update
2 **forever do**
3 $(a, c) \leftarrow observe(\Psi)$ ▷ Obtain a new event from stream Ψ
4 **if** $analyze((a, c))$ **then**
▷ Obtain the replayer status for the given process instance
5 $(state, cost, time) \leftarrow M_O(c)$
6 **if** $(state, cost, time) = \perp$ **then**
▷ There is no replayer associated, a new one is needed
7 $M_O(c) \leftarrow (state, cost, time) \leftarrow (s_i, 0, now)$
8 **end**
▷ Fetch the transition to follow (OCTSs are deterministic: only one transition is labeled a)
9 $(state, a, new\text{-}state) \in \{(d_s, \sigma, d_t) \in \delta \mid d_s = s_i, \sigma = a\}$
10 $M_O(c) \leftarrow (new\text{-}state, cost + w((state, a, new\text{-}state)), now)$ ▷ Replay
▷ Cleanup the map, removing old elements
11 **if** $|M_O| > m$ **then**
12 $R \leftarrow \text{argmin}_{(c,s,c,t) \in M_O}\{t\}$ ▷ Get oldest elements in M_O
13 Remove R from M_O
14 **end**
15 **end**
16 **end**

alignment can be provided for the latest events observed. Please note, however, this alignment might be sub-optimal.

3.2 Online conformance

Given an OCTS, the actual online conformance procedure is reported in details in Alg. 2. Specifically, the algorithm expects an OCTS and an infinite event stream as input, as well as the maximum number of process instances that we expect to have in parallel. Then, the algorithm constructs a hash map M_O (line 1) which, given the case id of an observed event, returns a tuple containing, for that specific instance, the state of the OCTS reached so far, the deviations cost until now, and a numerical representation of the last update (e.g., the Unix timestamp). The algorithm, then, begins the actual online procedure by repeating forever (line 2) the main loop which consists of the observation of a new event from the stream (line 3), a decision whether the event has to be analyzed or not (line 4) and the actual analysis (lines 5-14).

The analysis starts obtaining the state of the current process instance from M_O (line 5). If the map does not contain information (e.g., because this is the first event with this case id ever observed) then a new process instance is assumed (line 7). After that, the algorithm computes which is the next state reachable with the lowest cost (line 9) and updates the status of the current process instance with the new state, new total cost, and last update time (line 10). The last operation performed by the algorithm is a cleanup: this is necessary to keep the memory bounded and consists in dropping all states referring to the old executions (lines 12-13). This is necessary to keep the memory bounded and to avoid that an infinite stream causes an infinite memory usage.

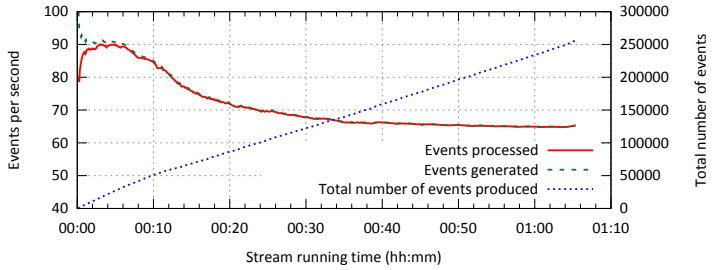


Fig. 3: Performance evaluation: events generated versus the events processed.

Please note that the operations not requiring constant time are the selection of the next state (line 9) and the removal of old elements from M_O (line 12). Since OCTS is a deterministic transition system, the former has complexity linear on the alphabet size (i.e., the number of transitions in the original Petri net), which is assumed to be constant over time. The latter has linear complexity on the size of M_O which, again, is constant over time. Therefore, the theoretical computational complexity of the algorithm makes it a viable solution for online applications.

4 Implementation and Results

The entire approach has been implemented[5] in the process mining toolkit ProM. As for other online plugins, the current implementation connects to a stream source that emits events (via a TCP connection) and processes each of them independently. This approach allows a strong decoupling between the source of the events and the actual online process mining tool.

In order to assess the feasibility of our approach, we simulated the stream of a process model comprising 26 tasks, and 20 gateways using PLG2 [8][6]. We have been able to simulate an unlimited event stream, with events referring to the given model. Specifically, we configured PLG to generate up to 90 events per second. Then, we tested the capabilities of our implementation, by running the conformance checker for about 1 hour and 10 minutes. As plotted in Fig. 3, after this period of time, 256110 events were generated. The generator, however, was not capable of keeping the given pace (we used a standard office laptop machine), and the system stabilized in simulating about 65 events per second. As the chart reports, all generated events were processed in time by our prototype. This demonstrates the actual feasibility of the approach, even in prototypical implementations.

[5] The implementation is available at `https://andrea.burattin.net/public-files/online-conformance/source.zip`. It will be moved into the ProM repository.

[6] The BPMN model is available at `https://andrea.burattin.net/public-files/online-conformance/model.pdf`.

(a) Main dashboard of the plugin (b) Routing of frequent errors

Fig. 4: Screenshots of the ProM plugin implementing the approach.

The ProM plugin implemented comprises a "dashboard" (cf. Fig. 4a) which shows, on its left-hand side, the running process instances (color-coded by severity and sorted by update time or by severity). The right-hand side of the dashboard contains system charts with number of errors every 5 seconds, number of processed events per second, number of traces in memory, and total memory consumption. The second component (cf. Fig. 4b) reports the frequent deviations on top of the behavioral model.

5 Conclusion and Future Work

This paper presents the first approach to compute conformance checking for online data streams. The fundamental advantage of this technique, with respect to previous off-line approaches, is the ability to check deviations from the reference behavior in real-time, i.e., immediately after they occurred. This way, possible corrections can be immediately enacted. The input of the presented technique is a Petri net, which is converted into a transition system. Such transition system is decorated with additional arcs in order to allow for deviations. Non-zero costs are associated with transitions representing deviations. Behavioral properties are employed to detect the target state of deviating transitions. The whole approach has been implemented in ProM.

We plan to continue the work presented on this paper by improving the conversion from Petri net to transition system using more conformance-oriented techniques as well as by exploiting contextual information (e.g., data associated with states).

Acknowledgements. We would like to thank Jorge Munoz-Gama for discussing early stage ideas of the approach. This work was partially funded by the Spanish Ministry for Economy and Competitiveness (MINECO) and the EU (FEDER funds) under grant COMMAS (TIN2013-46181-C2-1-R).

References

1. van der Aalst, W.M.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin / Heidelberg (2011)
2. van der Aalst, W.M., Adriansyah, A., van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. Adriansyah, A.: Aligning observed and modeled behavior. Phd thesis, Technische Universiteit Eindhoven (2014)
4. Aggarwal, C.C.: Data Streams: Models and Algorithms, Advances in Database Systems, vol. 31. Springer US, Boston, MA (2007)
5. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis Learning Examples. Journal of Machine Learning Research 11, 1601–1604 (2010)
6. vanden Broucke, S., Munoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J.: Event-based Real-time Decomposed Conformance Analysis. In: Proceedings of Confederated International Conferences: CoopIS. pp. 345–363 (2014)
7. Burattin, A.: Process Mining Techniques in Business Environments, LNBIP, vol. 207. Springer International Publishing (2015)
8. Burattin, A.: PLG2 : Multiperspective Process Randomization with Online and Offline Simulations. In: Proceedings of the BPM Demo Track. CEUR-WS.org (2016)
9. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online Discovery of Declarative Process Models from Event Streams. IEEE TSC 8(6), 833–846 (2015)
10. Burattin, A., Maggi, F.M., Cimitile, M.: Lights, Camera, Action! Business Process Movies for Online Process Discovery. In: Proceedings of TAProViz (2014)
11. Burattin, A., Sperduti, A., van der Aalst, W.M.: Control-flow Discovery from Event Streams. In: Proceedings of IEEE CEC. pp. 2420–2427. IEEE (2014)
12. Gaber, M.M., Zaslavsky, A., Krishnaswamy, S.: Mining Data Streams: a Review. ACM Sigmod Record 34(2), 18–26 (jun 2005)
13. Gama, J.: Knowledge Discovery from Data Streams. Chapman & Hall/CRC (2010)
14. Golab, L., Özsu, M.T.: Issues in Data Stream Management. ACM SIGMOD Record 32(2), 5–14 (jun 2003)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Software & Systems Modeling pp. 1–33 (2016)
16. Maggi, F.M., Burattin, A., Cimitile, M., Sperduti, A.: Online Process Discovery to Detect Concept Drifts in LTL-Based Declarative Process Models. In: On the Move to Meaningful Internet Systems. pp. 94–111. Springer Berlin Heidelberg (2013)
17. Maggi, F.M., Montali, M., van der Aalst, W.M.: An operational decision support framework for monitoring business constraints. In: Proc. of FASE (2012)
18. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.: Monitoring Business Constraints with Linear Temporal Logic : An Approach Based on Colored Automata. In: Proceedings BPM. pp. 132–147. Springer (2011)
19. Munoz-Gama, J.: Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. Springer (2016)
20. Munoz-Gama, J., Carmona, J., van der Aalst, W.M.: Conformance checking in the large: Partitioning and topology. In: Proc. of BPM. pp. 130–145. Springer (2013)
21. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
22. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
23. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Event stream-based process discovery using abstract representations. Knowl Inf Syst (May 2017)